CONTENTO: An open simulator for storage and content management services in mobile edge computing

Rafael Pérez-Torres
School of Computer Science and Information Technology
University College Cork
Cork, Ireland
r.perez@cs.ucc.ie

Thuy Truong
OCTO Research Office
Dell Technologies
Cork, Ireland
thuy.truong@dell.com

Donagh Buckley
OCTO Strategy Office
Dell Technologies
Cork, Ireland
donagh.buckley@dell.com

Cormac J. Sreenan
School of Computer Science and Information Technology
University College Cork
Cork, Ireland
cjs@cs.ucc.ie

***Abstract*—As the research on edge computing rises, tools are required to assess how the workloads from users impact on the resources of mobile networks and how the edge and its storage platform can further handle those workloads to improve the perceived Quality of Experience (QoE). Nevertheless, simulators for cloud and edge computing have mostly centred on the computing resources to conduct research on high performance computing, leaving storage aside. Thus, there is a lack of simulation tools focused on storage resources and how content is handled across the network. Storage is a valuable resource that can be exploited in storage services, like edge Content Delivery Networks (CDNs) to reduce delivery latency and the use of backhaul links. Thus, research on this field is compelling to improve network and applications performance, and specialized tools can promote it. Here, we present a Content OrieNTed Edge computiNg simulaTOr (CONTENTO) for the design of content management strategies through a customizable processing pipeline. CONTENTO allows edge applications to use the pipeline for tasks including content popularity detection and IoT data processing, and collects data rate and traffic statistics to assess the management strategies. We demonstrate CONTENTO's flexibility for implementing and experimenting with relevant case studies for the uplink and downlink. CONTENTO contributes by providing a common ground to define content management techniques and evaluate their performance under varying workloads and network configurations.**

***Index Terms*—simulator, storage services, edge computing, event-driven, content management**

I. Introduction

Simulators are compelling tools to design resource management systems for cloud and edge computing and to evaluate network's performance under different workloads from users, without deploying a real-world infrastructure. As edge computing approaches like Multi-Access Edge Computing (MEC) become a reality [1], [2], there is an increasing need for simulators to measure how the computing, storage, and communication resources of the mobile network are exploited. For instance, there are simulators focused on the use of computing resources [3]–[5] that produce indicators like the number of Virtual Machines (VMs) and processors utilised, computing tasks delay, the efficiency of load balancing strategies, and further measures for high performance computing.

Nevertheless, there is a lack of tools focused on storage and content handling across the network. With the increasing storage provided by edge computing, the path is open for novel storage services [6] both in the uplink and downlink, such as edge Content Delivery Networks (CDNs) for multimedia streaming, and IoT data processing applications, respectively. These services are powered by content management strategies including content caching, prefetching, and IoT data processing, among others. Thus, novel research can study storage-related issues like how content management strategies affect the delay of requests from user equipment (UEs) [7], optimizing the use of backhaul links, and enabling the incremental processing of IoT data from devices such as vehicles and sensors [8], to name a few. However, research in these directions is difficult if simulation tools specialised on content management are not available.

To tackle the issue, we present a Content OrieNTed Edge computiNg simulaTOr (CONTENTO) focused on the design and evaluation of content management strategies. To this end, we enabled CONTENTO with key features including: named content, a network topology with a hierarchy of hosts, mobility of UEs over base stations, independent links and realistic content transmission, and support for injecting strategies for content management. Although some simulators have support for a subset of these features, there is no tool supporting them all. Particularly, named content and its transmission are not part of their design features: content is seen just as an anonymous payload for the input/output of computing tasks. As a result, adjusting these computing-focused simulators towards a content-oriented approach is difficult, as many of their core features are not related to content management and must be extracted away. Unlike existing tools, CONTENTO implements all the mentioned features focusing on the edge, providing a flexible framework to integrate new solutions to

This publication has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) and is co-funded under the European Regional Development Fund under Grant Number 13/RC/2077.

manage the content and storage across the network. Furthermore, to demonstrate CONTENTO's flexibility, we present edge CDN and IoT data processing as relevant case studies and describe how our tool can simulate them.

Overall, CONTENTO helps to investigate the following research questions:

- What content management strategies can efficiently use edge storage resources? As we define a processing pipeline that can be customized to decide how content is handled. For example, we provide elements to detect content popularity for content caching.
- What is the impact of edge and network configuration on the produced traffic and transmission rates? As we allow to configure network's topology (hosts and links).
- What is the impact of changes in the request workload on the produced traffic and transmission rates over the edge? As we allow to define the input workload for both uplink and downlink scenarios.
- How to create strategies for the processing of UEs' data using edge storage? As the processing pipeline also handles uploaded data, edge storage can be leveraged to defer data processing according to application requirements.

Thus, our contribution is an open and flexible simulator that provides a common ground to study how content is transmitted across the network under defined content management techniques.

The rest of this article is structured as follows. Section II presents related work in simulators. Section III introduces CONTENTO's design and features, followed by case studies in Section IV. Experimentation on case studies are presented in Section V. Future work is described in Section VI, followed by conclusions in Section VII.

II. Related work

Most of existing simulators for cloud and edge computing focus on the performance of computation offloading strategies like load balance in terms of latency and overall CPU usage. Arguably, the most popular simulator is CloudSim [3], which is focused on simulating the execution of computing tasks, denominated cloudlets, that are offloaded to VMs run by hosts in remote data centres. Nevertheless, CloudSim implements a flow network model [9] on which packets are anonymous (they do not refer to files or named entities) and the only information collected is their size, source, and destination hosts. Similarly, in CloudSim the data transmission is simulated at a very high level, as packets are transmitted as a whole, and the transmission times are calculated simply as the $data\ size/bandwidth$. Although this makes sense on computation offloading as tasks are not executed until the full payload is received, for content transmissions this does not apply. Additionally, as CloudSim focuses on cloud computing, the definition of a mobile network architecture (that is, edge servers, base stations, UEs, and corresponding uplinks and downlinks) is out of its scope.

Given the popularity of CloudSim, forks have been developed to fit other scenarios. Among them, we can find CloudSimPlus [4], iFogSim [10], and EdgeCloudSim [5] (a comprehensive review of simulators can be found in [11]). Although these tools include some features for edge scenarios (e.g., EdgeCloudSim supports mobility of UEs), they still lack support for named content, base stations, and configurable content handling. Such elements are fundamental to study content management in mobile networks.

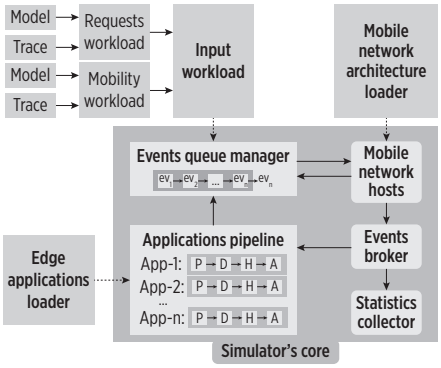


Fig. 1: CONTENTO's architecture.

Similarly, simulators like NS2 [12] and NS3 [13] are focused on the low-level details of internet transmissions. Although these tools might provide a more realistic simulation of network transmissions, they are not designed for high level content management.

Thus, it is possible to concentrate on the high-level functionalities required for content-oriented simulation including a) what content is requested, b) when content is requested, and c) where content is placed or pushed, rather than modelling features not related to content management, such as virtual machines and processors, or the full-stack implementation of transmission protocols. As a result, we identify the need for a simulator with content handling as a core design feature to study network's performance in terms of storage resources use.

III. CONTENTO's design

In order to support content management techniques in the uplink and downlink, we enabled CONTENTO with key features for simulation-related (mobile network architecture, event-driven design, flexible input workload, and statistics generation) and storage-related functionalities (edge applications, named content and content types, and flexible processing pipelines).

CONTENTO uses the architecture shown in Fig. 1and requires as input a mobility and request workload, a network architecture, and individual processing pipelines for edge applications. As CONTENTO is an event-driven simulator, every simulation step corresponds to the processing of an event (described later), which has origin and destination hosts in the network architecture. Broadly speaking, the events can refer to the transmission of different types of content, UEs movements, and incidences detected by applications. Before simulation, CONTENTO creates the initial events to handle the workload and connects the applications with the hosts (broker) to allow the reception of events. During simulation,

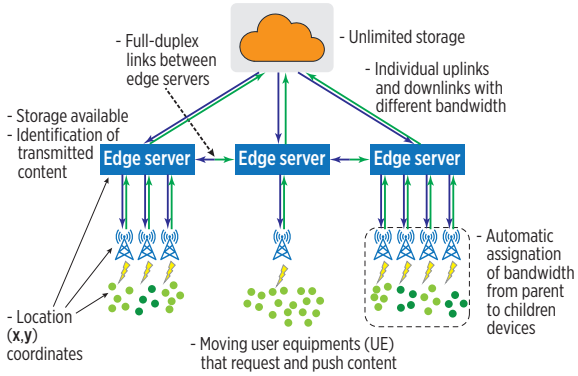


Fig. 2: Hosts and links of the supported network architecture by CONTENTO.

CONTENTO directs events to the corresponding application's processing pipeline, simulating how applications react to the input workload. As output, CONTENTO produces statistics for the delay and traffic and the output log of the processing pipeline. We describe CONTENTO's key features in the following paragraphs.

A. Simulation-related features

1) Support of a mobile network architecture: CONTENTO allows to customize a network architecture over an area of configurable size. As shown in Fig. 2, the network is logically defined as a tree, and the supported hosts are as follows:

- Cloud: we assume a single cloud host with unlimited storage space and where files are always available.
- Edge servers: with storage capabilities and individual uplinks and downlinks for connection with the cloud and children base stations.
- Base stations: connecting UEs with edge servers to download and update content. Base stations have spatial coordinates and a range, which are used to define the UEs connected to them. Base stations do not have storage or management capabilities, they only relay transmissions.
- UEs: which can roam and cause connections and disconnections from base stations. We automatically handle such changes, so that the input mobility is only a location trace (i.e., the input mobility does not require to know the number and distribution of base stations).

Cloud and edge servers have a file system to store content and an API for common operations (store and delete), which are exploited by the content management strategies.

We model independent uplinks and downlinks with configurable capacity. Currently, each parent seeks to divide the wireless channel capacity equally amongst children. For base stations, we automatically handle the changes in allocated link capacity caused by UEs' handover. Also, CONTENTO supports that edge servers are optionally inter-connected using a dedicated and configurable link, useful to push content and share events. To ease architecture definition, CONTENTO automatically distributes base stations in a hexagonal pattern

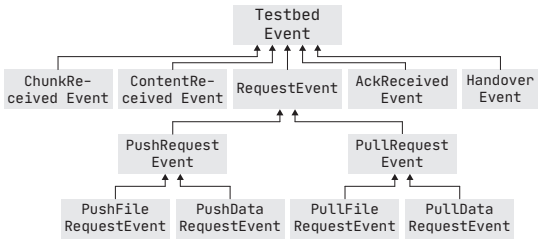


Fig. 3: The hierarchy of events employed by CONTENTO.

(based on their range), but it is possible for users to provide a list of them with their locations.

2) Event-driven design: CONTENTO is an event-driven simulator [14], so it works exclusively by sending message events between entities. Events are incidences that unfold across time, and the referred entities are the hosts in the network. An event e is a tuple $e = (id, type, src, dst, t)$, consisting of an identifier, a type, a source and destination host, and a timestamp. CONTENTO holds a queue of events (sorted by timestamp in ascending order), and at each step the first event is extracted and processed by its destination host. Hosts can add events to the queue, so that the simulation continues while pending events exists. Figure 3 shows the hierarchy of defined events, which can be described as follows.

a) Chunks transmission and link capacity use: A chunk is a portion of a file (e.g. a video segment), or a whole file of a small size. For a pair of connected hosts, A and B, the host A posts a `ChunkReceivedEvent` to schedule the reception of a chunk at host B. The event's timestamp is calculated by dividing the chunk size by the available capacity of the link, adding the propagation delay and a constant processing delay. Then, the host A reduces the available capacity in the link with B. When the chunk is received at B, it restores the capacity and posts an `AckReceivedEvent` for host A, so that it can continue the transmission of pending chunks.

b) Content transmission (as a whole): We produce a `ContentReceivedEvent` when all the chunks of a content have been received by a host. It is possible that an edge server does not receive all the chunks for a content as UEs can move to base stations in other servers during transmission. In any case, CONTENTO notifies this event to applications' pipelines to execute a content management strategy.

c) UEs' mobility: The `HandoverEvent` indicates that a handoff has occurred, that is, a UE is now serviced by a different base station. This event causes adjustments to keep current transmissions and for UEs' bandwidth from base stations, as the base station being left will have more link capacity to share with UEs, and the new base station will have less capacity for connected UEs. In any case, this event is automatically created and handled.

d) Content requests: Finally, the `RequestEvent` class hierarchy models content requests, which are typified as: `PullRequestEvent` for the typical requests for content (e.g. file downloads), and `PushRequestEvent` for the asynchronous transmissions started by a data source UE (e.g.

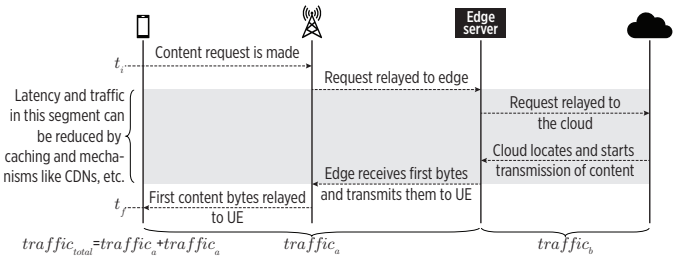


Fig. 4: The latency and traffic reduction achievable through content management techniques like caching and prefetching.

an IoT device pushing data in the uplink).

3) Input workload: The input workload consists of the UEs' mobility and content requests. While UE's mobility is application-independent (accessible for all applications), requests are associated with an edge application. Since CONTENTO automatically handles the mobility changes of UEs, trajectories from mobility models or existing datasets can be employed as input. Currently, CONTENTO embeds the Random Waypoint algorithm to generate mobility.

Regarding requests, `PullRequestEvents` and `PushRequestEvents` are allowed at the uplink and downlink. The following pull requests can be simulated:

- UE-to-Cloud requests: representing the typical content downloads by UEs in web browsing or media streaming.
- UE-to-UE requests: e.g. UEs accessing data produced by another UE (e.g., when IoT data is remotely queried).
- Cloud-to-UE: accounting for remote applications/web services requesting IoT data produced by a UE.

CONTENTO features a basic content generator, which can create a set of files of a random size (number of files, minimum and maximum size are configurable).

In CONTENTO, push requests represent the asynchronous data transmissions from UEs, required for IoT scenarios where data is pushed and processed at the edge. Each request can configure whether pushed data should reach the edge or the cloud.

Notice that additional push and pull requests between edge servers and the cloud are generated when processing the workload according to the injected content management strategies.

4) Statistics for latency and traffic: CONTENTO collects statistics for the traffic and delivery latency produced by the injected content management strategies. As shown by Fig. 4, improvements can occur in the cloud-edge segment if content is cached at the edge, as the roundtrip from the cloud is avoided. Specifically, CONTENTO collects data from each chunk, including origin, destination, size, transmission times, and associated request. From such data, the traffic in network segments and the transmissions rates are produced.

B. Storage-related features

1) Edge applications: In edge computing, there could be several running edge applications sharing the servers resources through virtualization. As our focus is on content and storage,

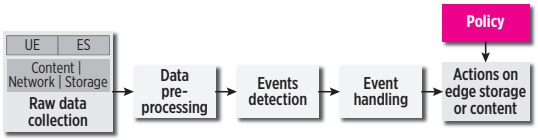


Fig. 5: The processing pipeline to create content management strategies.

CONTENTO helps to define edge applications with a storage quota and a processing pipeline. The quota is the storage available for an application across all edge servers, that is the sum of used space in edge servers is equals or lower than the quota. The processing pipeline (described later) tailors the detection of relevant events and produced actions, creating strategies for content management.

2) Named content: CONTENTO supports named content, i.e., content identifiable through a URI. Named content is vital in storage management, as techniques must know the specific content being handled. As hosts produce heterogeneous content, we define the following content hierarchy:

- File: typical files accessible through a URI (e.g. www.server.com/file1.txt), initially located in the cloud. Files are changeless, so they can be cached at edge servers.
- Data: produced by UEs (e.g. IoT data). We define a URI for it based on the path to its source UE (e.g. Cloud/ES0/BS1/UE4/video). As data can be produced continuously, we assign a timestamp to represent its freshness.
- Message data: control messages sent between hosts, only used for internal control (never seen by applications).

Notice that regardless of its type, content holds a size attribute.

3) Flexible content processing pipeline: On top of the low-level content transmissions, CONTENTO lets edge applications specify a processing pipeline to detect relevant events and define content management strategies. For security and privacy, pipelines are application-specific, i.e., an application cannot access data from other applications. The processing pipeline, shown in Fig. 5, is executed at edge servers and the cloud, and it targets content as follows:

- Data and file contents from pull requests: for typical content downloads in the downlink and uplink, allowing to detect events like content popularity.
- Data from push requests: for uplink transmissions, targeting the data pushed by UEs and allowing to execute application-defined techniques.

We briefly review the stages of the processing pipeline according to the handled request type.

a) Processing pipeline for pull requests: CONTENTO uses the `ContentReceivedEvent` to infer the metadata of a transmitted content (file name, size, etc.). The pre-processing stage for this request type is bypassed. In the event detection stage, the simulator accumulates the information of the received event, enabling the analysis of historic transmissions records for detecting popularity. The event handling specifies the actions for reacting to the event, which are executed in the actions stage. Policies can be specified as parameters for

executing actions. For example, an action can be defined to push a file and a policy can control its destination (e.g., parent, sibling, or children hosts).

b) Processing pipeline for push requests: In this case, the pipeline resembles the stages of pattern recognition techniques. First, metadata is extracted from the pushed data (in the `ContentReceivedEvent`). This is customizable, for instance, images could use metadata as height, width, resolution, etc. The pre-processing stage defines the techniques to prepare the data for event detection, e.g. image cropping, segmentation, etc. In the event detection stage, a classifier identifies an event relevant for the application (e.g., the presence of cars or people in the image). Finally, the action stage can notify the event to external subscribers or push it to edge servers (configurable via policy) to disseminate its information.

Notice that if pipeline elements are not specified, CONTENTO only simulates the content transmission.

4) Extensible features: CONTENTO is implemented in the Java platform, enabling dependency injection to customise content types and the processing pipeline. For instance, the `Data` class can be extended to define images, sound, or other types, and CONTENTO transparently handles the transmission. This is useful to model the data produced by IoT devices.

For the processing pipeline, we provide parameterised pre-processors, event detectors, and actions than can be combined and extended for content management and data processing. For example, we include a basic popularity detector to identify changes in content popularity (low to high popularity, and vice versa) using as parameters a time interval and a threshold requests number. Regarding actions, we include actions applicable to files (*Store* and *Push*), and data (*Notification*).

Also, additional statistics can be generated from the information of transmitted chunks, such as the most popular content for UEs, and the time with the highest requests, which are of high relevance to create context-aware storage management.

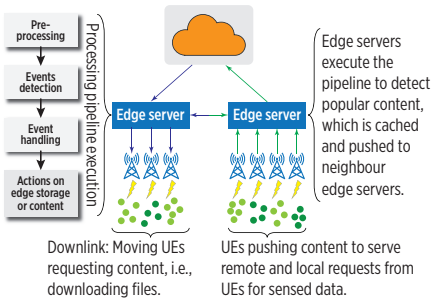
Finally, CONTENTO is a flexible yet lightweight tool that can be executed in regular desktop or laptop equipments.

IV. Case studies

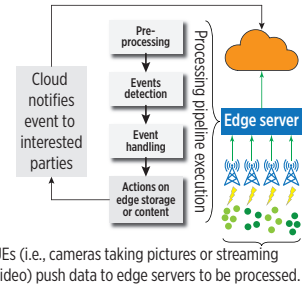
CONTENTO offers flexibility for scenarios focused at the uplink or downlink. As representative case studies, revisited in the experiments section, we present the following ones.

A. Edge CDN

The storage in edge servers can be exploited to create edge CDNs [15]. We can study how edge CDNs help to automatically cache the transmitted content, as shown in Fig. 6a, reducing the latency and energy consumption of hosts to serve further requests. In the uplink, such requests are those where the data is produced by UEs, for instance, remote UEs of a weather sensing system. In the downlink, requests refer to downloads by users while browsing the web. In any case, it is possible to use strategies based on content popularity to decide what content to cache at edge servers, and even to push it to neighbour servers as it might gain popularity in near areas.



(a) An edge CDN scenario aimed at uplink and downlink.



(b) An IoT data processing scenario for the uplink.

Fig. 6: Sample case studies for CONTENTO.

B. IoT data processing

The use of edge computing for IoT data processing has been researched for producing information (IoT big data [16]) and addressing the bottlenecks of cloud-based solutions. As shown in Fig. 6b, the data produced by UEs can be pushed to edge servers for its processing, which should be easy to adjust for application developers. In some scenarios, real-time processing might not be needed, so it can be scheduled for periods of low use of edge resources or when it is suitable for the application (e.g., at the end of the day). In these cases, the edge storage becomes vital to store the data until it is processed.

As a sample case study, we define a computer vision application for Automated License Plate Recognition [17]. In this application, cameras, i.e., fixed-location UEs, upload pictures of vehicles driving above the speed limit for detecting fast speed drivers. The pictures are processed by edge servers to detect car plate numbers, which are notified to endpoints of external subscribed systems. We defer (schedule) the processing, relaying on edge storage to store and queue pushed data. Additionally, the detected event is pushed to neighbour servers to disseminate it.

V. Experiments

We conducted experiments to demonstrate the use of CONTENTO for different use cases. Particularly, we evaluated the feasibility of using CONTENTO to model and execute the aforementioned case studies and its flexibility to use different architecture and workload parameters. Also, we assessed CONTENTO's scalability over a varying number of requests.

For these experiments, we based on the architecture shown in Figure 6a, adjusting it with the parameters in Table I. Al-

TABLE I: Simulation's parameter settings for the presented experiments (BS=base station, ES=edge server).

Area	Zone	10km x 10km
	BS range	1.8 km
	Generated BSs	20
	UEs	400, 500, 600
Storage	Number of ESs	2
	Storage per ES	10 TB
Links	Uplink capacity between cloud and ESs	1 Gbps
	Downlink capacity between cloud and ESs	1 Gbps
	Uplink capacityw between ESs and BSs	1 Gbps
	Downlink capacity between ESs and BSs	1 Gbps
	Uplink capacity between BSs and UEs	256 Mbps
	Downlink capacity between BSs and UEs	256 Mbps
Workload	Simulated time	12 Hours
	Available files	1000, 10000
	Simulated requests	12000
	Popularity policy	3 reqs. in last hour
	Minimum file size	1 KB
	Maximum file size	2 MB

though we set the base stations range to 1.8km (automatically creating 20 base stations for full coverage area), in practice it depends on attributes like the underlying network standard, rural/urban environment, etc. In any case, CONTENTO allows the range and other network and workload parameters to be configured. For simplicity, a single application is defined in each scenario.

A. *Edge CDN*

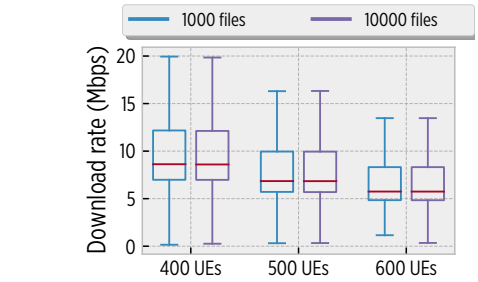
The edge CDN case study can be implemented through the following steps:

1) Define a network architecture over a geographical area.
2) Produce the mobility traces of UEs with a mobility model or using existing locations traces.
3) Define a request workload with each request made by a UE for a file in the cloud, including its timestamp.
4) Customise the processing pipeline to detect content popularity and cache content.

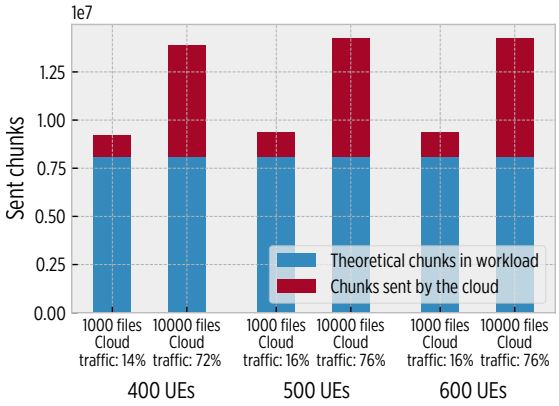
Steps 1 and 2 are supported by simply specifying the parameters for the area and architecture (links capacities, number of edge servers and UEs) and using the internal mobility generator. CONTENTO creates the required base stations to cover the zone and attach UEs to them. For step 3, we employ CONTENTO's file generator.

For step 4, the pipeline is customized as follows. First, a `PopularityEventDetector` (provided by our tool) is executed after any `ContentReceivedEvent` is generated. The `PopularityEventDetector` automatically produces a `HighPopularityEvent` if popularity is found under specified parameters. For that event, we set a `StoreAction` and a `PushAction` to be executed. The overall pipeline configuration is:

```
1 app.addEventDetector(PPOutcomePullRequest.class, new
      PopularityEventDetector(HOUR, 3));
2 app.addActionForEventType(HighPopularityEvent.class,
      new StoreAction());
3 app.addActionForEventType(HighPopularityEvent.class,
      new PushAction());
4 app.setPolicyForActionType(PushAction.class, new
     PushPolicy(false, true, false));
5 app.setOpportunisticStorage(true);
```



(a) Changes in data transmission rates as a function of the number of UEs in the network.



(b) Produced traffic under varying UEs.

Fig. 7: Data rate and traffic results for the edge CDN case study.

By doing so, we instruct CONTENTO to cache any content requested 3 or more times during the last hour and to push it to neighbour servers. The *opportunistic storage* flag indicates servers to cache all the transmitted contents; if popular content is detected and application's free space is low, non-popular contents are automatically deleted to make space.

For this experiment, we observed the following results. Regarding data transmission rates, Fig. 7a shows that the rates are lower when more UEs are present, which is caused by the less link capacity available for UEs. Regarding traffic, Fig. 7b shows the traffic (in terms of chunks) transmitted by the cloud using the simple popularity caching strategy. Notice that the saved traffic will depend on the cached content: when only 1000 files are available to download for the 12000 requests, more content is cached so the saved traffic is higher (avg 85%) than when 10000 files are available (avg 25%).

B. *IoT data processing*

We use the same four steps to simulate the plate number detection case study. First, we define a network architecture on which the road cameras are the UEs. Then, we specify the input workload: the mobility for UEs is disabled by specifying an empty location trace, and the requests define random push requests of camera data. In the final step, we specify the processing pipeline, but as this is a rather highly customized scenario, the next steps must be performed.

First, some classes are extended for this application's needs:

- `CameraData` extends `Data`: To define the picture's height, width, resolution, and location attributes.
- `PlateRecognitionPreProcessor` extends `PreProcessor`: To define the pre-processing tasks applied to the received picture. For the sake of simplicity, we just encapsulate the picture content in another object.
- `PlateRecognitionEvent` extends `ApplicationEvent`: To define the attributes of the application event, i.e. the observed plate number.
- `PlateRecognitionEventDetector` extends `BaseEventDetector`: To define the mechanism to detect the plate number. For the sake of simplicity, here we generate a random plate number.

Then, the processing pipeline can be specified. As this application notifies external subscribers about detected `PlateRecognitionEvents`, we set a `NotificationAction` as an action. A `PushAction` action is specified as well to disseminate events data to neighbour edge servers. Then, we specify a processing interval to defer the processing of uploaded images. The final configuration for this application is as follows:

```
1 app.addPreProcessor(ContentReceivedEvent.class, new
      PlateRecognitionPreProcessor());
2 app.addEventDetector(PPOutcomePlateRecognition.class
      , new EventDetectorPlateRecognition());
3 app.addActionForEventType(PlateRecognitionEvent.
      class, new NotificationAction());
4 app.setExternalSubscriberPerEventType(
      PlateRecognitionEvent.class,
5     List.of("http://www.policedepartment.com", "http
          ://www.fineticketingsystem.com"));
6 app.addActionForEventType(PlateRecognitionEvent.
      class, new PushAction());
7 app.setPolicyForActionType(PushAction.class, new
      PushPolicy(false, true, false));
8 app.setProcessingSchedule(PROCESSING_INTERVAL);
```

As shown in Fig. 8, CONTENTO is able to simulate the push requests from UEs, storing the data and triggering the processing at the configured time (43200 s). At that moment, CONTENTO executes the pipeline at edge servers to detect the plate numbers, pushing their data to neighbour edge servers, and advising the cloud to notify the external subscribers.

C. Simulation scalability

We evaluated CONTENTO's running times in a laptop equipment (Dell XPS 13 9380, with 16 GB of RAM, Intel Core i7-8565U processor @1.80GHz). As CONTENTO is event-driven, its performance is influenced by the number of events produced to handle the input workload.

A profiling of CONTENTO using IntelliJ IDEA reveals that chunks transmission dominates the running time. Nevertheless, as shown in Fig. 9, CONTENTO has a linear growth for its running times, simulating 50000 requests (and associated 3589 handover events) in around 133 seconds. Notice that since popularity is calculated by evaluating past requests, large workloads that undergo popularity detection would impact on the running time. Similarly, a high cache miss rate at edge servers causes the content to be transmitted from the cloud, which creates more internal events and increases the running time.

VI. Future work

We look to enable coordinated storage management techniques between edge servers to account for the status of storage resources and content popularity in other locations, using the cloud as a broadcaster. Such techniques must be parameterizable, so that applications can tailor how the information from neighbour servers impacts on content management.

Similarly, we are interested on features for more robust content management and data processing techniques based on the context information (e.g., content popularity and UEs' mobility [18]) already available for the network. For instance, the freshness of IoT data should be considered to decide whether requests are served with data cached at the edge or if an update from the UE is needed.

Additionally, although CONTENTO accepts traces as input, we are interested on integrating mobility and request models to provide a fully integrated environment to quickly generate and evaluate specific scenarios.

Finally, for IoT data processing, we will work to support actual files in the pipelines rather simulated content. For instance, the plate recognition scenario can use actual images from disk, employing machine learning techniques to evaluate how they would perform in an edge environment. To do so, we must integrate some high-level computing aspects, such as the required instructions per task and available computing power, without overseeing that our focus is on the storage resources.

VII. Conclusions

Despite storage is a valuable resource in edge computing, existing simulators are mostly focused on the computing capabilities of edge servers, disregarding how content management could impact on edge performance. To tackle the issue, we presented CONTENTO, a simulator specialised on storage resources and studying content management in edge computing scenarios. CONTENTO helps users to define mobile network architectures, edge applications, and workloads composed by mobility and content requests. CONTENTO allows applications to create content management strategies through a processing pipeline. Such strategies can operate on the data pushed by UEs in IoT scenarios (uplink), or study on the content downloads to detect popularity and further features for cache or prefetching techniques (downlink). Through representative case studies, we showed how CONTENTO is flexible enough to be extended and adjust to different scenarios, helping to study the impact of the workload on network performance (and vice versa) whilst using the application-defined processing pipelines. Thus, CONTENTO contributes to answer research questions by providing a common ground to define content management techniques and quickly evaluate their performance under varying workloads and network configurations. The source code for CONTENTO as an open source software is available at MISL website (http://www.cs.ucc.ie/misl/research/software/).

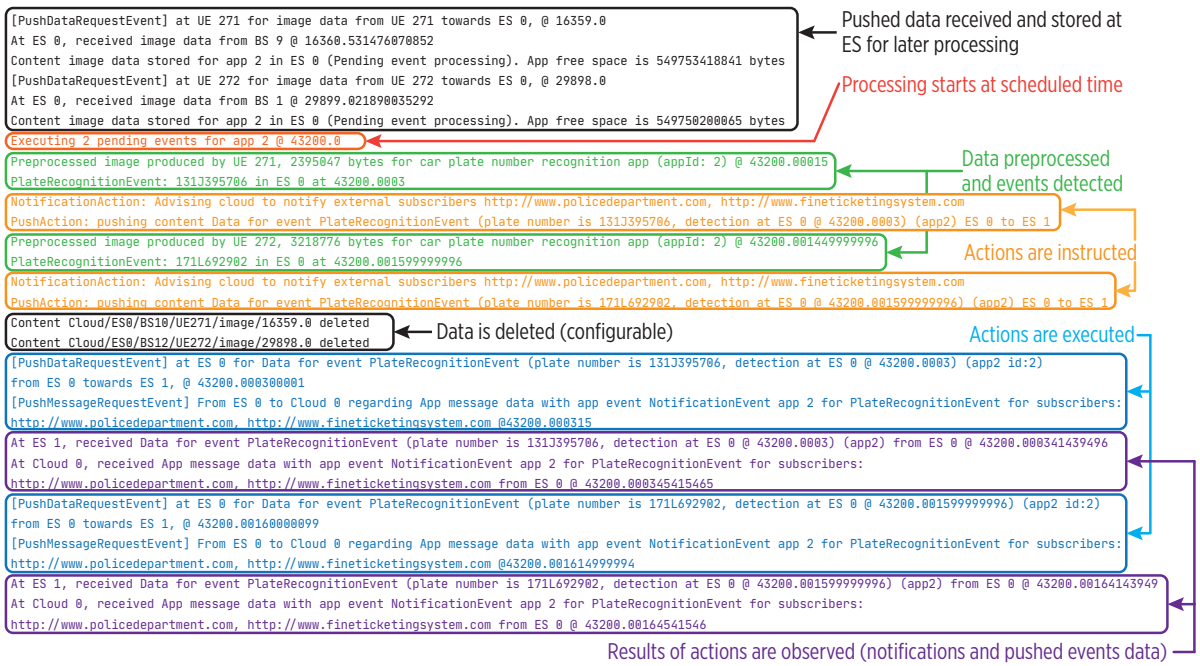


Fig. 8: Output of the processing pipeline for the plate recognition case study.

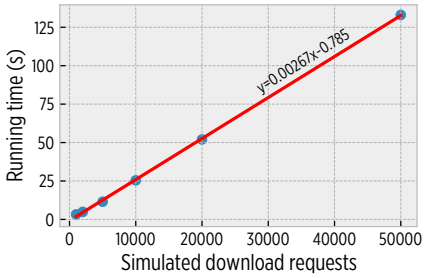


Fig. 9: Running times as a function of simulated requests.

References

[1] E. Ahmed and M. H. Rehmani, "Mobile edge computing: Opportunities, solutions, and challenges," *Future Generation Computer Systems*, vol. 70, pp. 59 – 63, 2017.

[2] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, Feb 2018.

[3] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, p. 23–50, Jan. 2011.

[4] M. C. Silva Filho, R. L. Oliveira, C. C. Monteiro, P. R. M. Inácio, and M. M. Freire, "Cloudsim plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, May 2017, pp. 400–406.

[5] C. Sonmez, A. Ozgovde, and C. Ersoy, "Edgecloudsim: An environment for performance evaluation of edge computing systems," in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, 2017, pp. 39–44.

[6] Y. Cui, Z. Lai, and N. Dai, "A first look at mobile cloud storage services: architecture, experimentation, and challenges," *IEEE Network*, vol. 30, no. 4, pp. 16–21, July 2016.

[7] J. Poderys, M. Artuso, C. Michael Oest Lensbol, H. Lehrmann Christiansen, and J. Soler, "Caching at the Mobile Edge: A Practical Implementation," *IEEE Access*, vol. 6, pp. 8630–8637, 2018.

[8] X. Sun and N. Ansari, "Dynamic Resource Caching in the IoT Application Layer for Smart Cities," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 606–613, apr 2018.

[9] S. K. Garg and R. Buyya, "Networkcloudsim: Modelling parallel applications in cloud simulations," in *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, Dec 2011, pp. 105–113.

[10] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[11] A. Markus and A. Kertesz, "A survey and taxonomy of simulation environments modelling fog computing," *Simulation Modelling Practice and Theory*, vol. 101, p. 102042, 2020, modeling and Simulation of Fog Computing.

[12] Information Sciences Institute, "The Network Simulator - ns-2," 2011. [Online]. Available: http://nsnam.sourceforge.net/wiki/index.php

[13] Nsnam, "ns-3 — a discrete-event network simulator for internet systems," 2020. [Online]. Available: https://www.nsnam.org/

[14] T. Faison, *Event-Based Programming: Taking Events to the Limit*, 1st ed. USA: Apress, 2011.

[15] R. Viola, A. Martin, M. Zorrilla, and J. Montalbán, "Mec proxy for efficient cache and reliable multi-cdn video distribution," in *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, June 2018, pp. 1–7.

[16] X. Cheng, L. Fang, X. Hong, and L. Yang, "Exploiting mobile big data: Sources, features, and applications," *IEEE Network*, vol. 31, no. 1, pp. 72–79, January 2017.

[17] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, ser. SEC '17. New York, NY, USA: Association for Computing Machinery, 2017.

[18] A. Elgazar, K. Harras, M. Aazam, and A. Mtibaa, "Towards intelligent edge storage management: Determining and predicting mobile file popularity," in *2018 6th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, March 2018, pp. 23–28.